Performance Evaluation and Comparison of Service-based Image Processing based on Software Rendering

Ole Wegen
Hasso Plattner Institute,
Faculty of Digital Engineering,
University of Potsdam, Germany
ole.wegen@student.hpi.de

Matthias Trapp
Hasso Plattner Institute,
Faculty of Digital Engineering,
University of Potsdam, Germany
matthias.trapp@hpi.de

Jürgen Döllner
Hasso Plattner Institute,
Faculty of Digital Engineering,
University of Potsdam, Germany
juergen.doellner@hpi.de

Sebastian Pasewaldt
Digital Masterpieces GmbH
Germany
sebastian.pasewaldt@digitalmasterpieces.com

ABSTRACT

This paper presents an approach and performance evaluation of performing service-based image processing using software rendering implemented using Mesa3D. Due to recent advances in cloud computing technology (w.r.t. both, hardware and software) as well as increased demands of image processing and analysis techniques, often within an eco-system of devices, it is feasible to research and quantify the impact of service-based approaches in this domain w.r.t. cost-performance relation. For it, we provide a performance comparison for service-based processing using GPU-accelerated and software rendering.

Keywords

cloud-rendering, image processing, software rendering, performance comparison

1 INTRODUCTION

1.1 Motivation

Today, image processing is a common task with various applications ranging from processing high-quality professional content to User-Generated Content (UGC). Within recent years, simultaneous developments with respect to the following major directions can be observed: (1) an increase of mobile hardware and processing capabilities, (2) an increase in cloud-processing capabilities and infrastructure, as well as (3) the expected increase of network throughput and infrastructure, with new transmission standards such as 5G and support for mobile devices such as Google Cloud Messaging (GCM).

There are numerous applications to service-based provisioning of image processing functionality, both Business-to-Customer (B2C) and Business-to-Business (B2B) application scenarios. Besides the protection or Digital Rights Management (DRM) of processing functionality internals or software source code [12], the most prominent is the integration of such services into collaborative web-based [2] and mobile applications as well as cloud-processing services. However, such a service-based provision has to account for two major aspects:

GPU-based Processing: For efficient processing, image processing components often rely on hardware-acceleration based on Graphics Processing Units (GPUs), thus require dedicated graphics hardware (GPU). This is often the case for real-time or high-performance applications. It should be taken into account that not only the actual processing time influences whether an application is real-time capable. Also the network speed needs to be considered.

Scalability: Providing image-processing functionality via cloud-processing services to end users relies on scalability features offered by cloud-service providers. However, currently dedicated GPU-instances are costly, thus infrastructures do not scale well for usability in the B2C market.

Taken both of the aspects into account when providing service-based image processing for customers, the financial impact varies significantly between hosting a dedicated GPU-based server or rely on scalable GPU-based cloud-computing services such as Amazon AWS Elastic Compute Cloud (EC2) (ranging from approx. €120 for a dedicated server to $2000 for a scalable one). However, one important aspect of service-based provisioning of any functionality in general, is the coverage of operational cost for the server infrastructure. With respect to this, one observation that can be made is the cost span between server with and without dedicated GPU hardware. To reduce costs while maintaining scalability simultaneously, *Software Rendering (SWR)* is a promising alternative to servers and services that support dedicated GPUs.

1.2 Problem Statement

Compared to GPU-based rendering, the run-time performance of SWR is exspected to be significantly inferior. Thus, especially for high spatial resolutions of input images, SWR is not suitable for applications that require fast, on-demand results, but it can be used for processing tasks where speed or output quality plays not a crucial role, e.g., for batch processing or preview generation. This work relies on an image-processing framework based on C++ and OpenGL that is an integral component of various desktop and mobile image-abstraction applications; given an input image and a description of an image-processing operation, it computes a transformed output image using GPU-acceleration.

However, depending on (1) the *type of spatial processing technique*, i.e., pixel-based, neighborhood-based, or global operations, as well as (2) the choice of *implementation*, there are approaches and frameworks that can counterbalance some of the negative run-time performance impact. Nevertheless, when using SWR there are arguments — specific to software development aspects — to rely on standard rendering and graphics Application Programming Interfaces (APIs) for implementation of image processing techniques:

Standardization: Using standardized APIs backed by industry and research allows for fast adaption to new software and hardware technology as well as ease the integration of new processing algorithms and techniques.

Software Maintenance: Software maintenance effort and costs can be lowered by relying only on a single framework for image-processing.

1.3 Approach and Contributions

This paper approaches the challenge of enabling SWR for image processing as follows: first, it integrates software rendering by using GPU emulation via The Mesa 3D Graphics Library (Mesa3D). This allows for a wide support of cloud-computing providers and thus facilitates vertical (adding processing power) and horizontal (adding computing instances) scaling. Based on this integration approach, performance measurements are obtained for comparing a dedicated server with GPU support and standard servers running the GPU-emulation.

The remainder of the paper is structured as follows. Section 2 reviews related work regarding service-based approaches for image and video processing systems and techniques. Section 3 describes the approach and implementation details for integrating hardware and software rendering. Section 4 presents and discusses results of a performance comparison between SWR and GPU-based processing in a service-based environment. Finally, Section 5 concludes this paper and discusses future research directions.

2 RELATED WORK

2.1 Software Rendering Approaches

Besides special approaches for the implementation of high-performance software rasterization using GPUs [18, 14], only a few research focus on software rendering in general.

Mileff and Dudra presents an overview of performance improvement methods for Central Processing Units (CPUs) by utilizing specific instruction sets and describe how these methods can be applied in (tile-based) software rendering [20]. Following this, the authors reviews problems and opportunities of two-dimensional rendering and propose and evaluate an efficient, software-based rasterization method for textures [21]. Mesa3D in particular, is used for in-situ visualization in a particle-based volume rendering Kyoto Visualization System [8].

2.2 Service-based Image Processing

Several software architectural patterns are feasible for implementing service-based image-processing [4, 25]. However, one prominent style of building a web-based processing system for any data is the service-oriented architecture [36]. It enables server developers to set up various processing endpoints, each providing a specific functionality and covering a different use case. These endpoints are accessible as a single entity to the client, i.e., the implementation is hidden for the requesting clients, but can be implemented through an arbitrary number of self-contained services.

Since web services are usually designed to maximize their reusability, their functionality should be simple and atomic. Therefore, the composition of services [10] is critical for fulfilling more complex use cases [16]. The two most prominent patterns for implementing such composition are *choreography* and

orchestration [26]. The choreography pattern describes decentralized collaboration directly between modules without a central component. The orchestration pattern describes collaboration through a central module, which requests the different web services and passes the intermediate results between them [28].

In the field of image analysis, Wursch *et al.* [40, 41] present a web-based tool that enables users to perform various image analysis methods, such as text-line extraction, binarization, and layout analysis. It is implemented using a number of Representational State Transfer (REST) web services and application examples include multiple web-based applications for different use cases.

Further, the viability of implementing image-processing web services using REST has been demonstrated by Winkler *et al.* [38], including the ease of combination of endpoints. Another example for service-based image-processing is Leadtools (https://www.leadtools.com), which provides a fixed set of approx. 200 image-processing functions with a fixed configuration set via a web API. In this work, however, a similar approach using REST is chosen, although with a different focus in terms of granularity of services.

Applications with respect to medical image processing are presented by Yuan *et al.* . [43] as well as Moulick and Gosh [22]. They propose a web-based platform to present and process medical images by using server-side computing for a series of image processing algorithms.

Further, in the field of geodata, the Open Geospatial Consortium (OGC) set standards for a complete server-client ecosystem. As part of this specification, different web services for geodata are introduced [23]. Each web service is defined through specific input and output data and the ability to self-describe its functionality[42]. In contrast, in the domain of general image-processing there is no such standardization yet. However, it is possible to transfer concepts from the OGC standard, such as unified data models. These data models are implemented using a platform-independent effect format [3]. In the future, it is possible to transfer even more concepts set by the OGC to the general image-processing domain, such as the standardized self-description of services.

2.3 Image Abstraction Techniques

In this work, we focus on edge-aware and content-preserving image-processing as a fundamental tool in computational photography and non-photorealistic rendering for abstraction and artistic stylization for application and testing purposes. Typical approaches that operate in the spatial domain for abstraction use a kind of anisotropic diffusion [27, 37] and are designed for parallel execution, such as approximated by the bilateral filter [35] and guided filter [9]. A plenitude of stylization techniques exist using these filters as building blocks to simulate traditional painting media and effects [13], such as cartoon [39] and oil paint [33]. However, these may become computationally expensive when applied in an iterative multi-stage process. This particularly applies to techniques using global optimizations to separate detail from base information, e.g., based on weighted least squares [6] or locally weighted histograms [11], and recent techniques that separate style from content using neural networks [7]. Because of their global optimization scheme, they are typically not suited for real-time application, in particular not on mobile devices. To this end, we implemented a variety of these techniques using the proposed image-processing service including stylization, High Dynamic Range (HDR) tone mapping and compression, JPEG artifact removal and colorization, to demonstrate its versatile application. We used a representative subset of these techniques for performance evaluation.

3 SOFTWARE RENDERING

This section describes the approach for enabling software rendering for an Open Graphics Library (OpenGL)-based rendering framework for image processing techniques [29]. In particular, based on our system requirements, this comprises justification of middle-ware choices and specifics for the deployment process that is suitable for cloud-computing providers.

3.1 Software Rendering using Mesa3D

A common approach for enabling software rendering for OpenGL-based applications [31] is using Mesa3D [19]: an open source 3D graphics library implementing OpenGL, Vulkan and other graphics API specifications. It offers multi-platform support and is used in Linux, Windows, and macOS. Basically, there are two architectures for Mesa3D driver implementation. The older one uses Mesa's Direct Rendering Infrastructure. Gallium3D represents the new architecture and API driver implementation and development by abstracting from specific hardware and operating systems. The following software drivers are available for Mesa3D:

SWRast: represent the original/legacy software rasterizer for Mesa3D implemented in C. It supports only the fixed-function rendering pipeline that was used in OpenGL before shaders were introduced.

OpenSWR: is a fast CPU OpenGL-compatible renderer developed by Intel [30] with advantages in cluster setups for large datasets.

Table 1: Comparison of Mesa3D software rendering drivers.

	Software drivers for software rendering			
Aspect	**SWRast**	**OpenSWR**	**Softpipe**	**LLVMpipe**
API Features	OpenGL 2.0 (no GLSL)	OpenGL 3.3 Core + OpenGL 3.0 Compatibility	OpenGL 3.3 (70.9% Extension Coverage)	OpenGL 3.3 (64.8% Extension Coverage)
Performance	Not tested	High	Low	High

Softpipe: is a full Gallium reference driver for a CPU supporting the programmable graphics pipeline by implementing code generation for shader programs and closely modeling of hardware behavior.

LLVMpipe: is a fast software rasterizer that supports a sufficient amount of OpenGL core functionality and extensions. It uses LLVM for x86 Just-in-Time (JIT) code generation and is multi-threaded. It is based on LLVM [17], which is a collection of modular and reusable compiler and tool chain technologies.

Since the framework is not limited to OpenGL[31], also Vulkan[32] should be supported by SWR, which unfortunately none of the available software renderer currently supports. For choosing a software rendering approach we have to account for the following aspects:

Level of Feature-Completeness: This depends on the *up-to-dateness* of a specific software renderer. Apart from SWRast, all available software renderer fully support OpenGL 3.3.; OpenSWR is currently more actively developed while Softpipe on the other hand supports additional OpenGL extensions. Nevertheless, also LLVMpipe supports a sufficient amount of extensions.

Processing Performance: The run-time performance of SWR approaches represents the most crucial aspect and is required being as high as possible.

Table 1 shows a comparison of the available Mesa3D software drivers, based on aggregated information obtained from the Mesa3D feature set [5]. The preliminary performance values are obtained using the approach described in Section 4. Specifically, the machine used for the preliminary test has the following hardware specifications: an Intel i5-8400, 2.8 GHz processor with six cores, and 16 GB Random Access Memory (RAM). Only a separated morphological closing operation with a kernel size of three was tested as representative for multi-pass processing techniques that are based on neighborhood sampling operations (Figure 2c). Figure 1 shows the resulting performance values at a logarithmic scale. It can be observed that Softpipe was significantly slower than LLVMpipe while OpenSWR was also slower than LLVMpipe but only by a smaller factor.

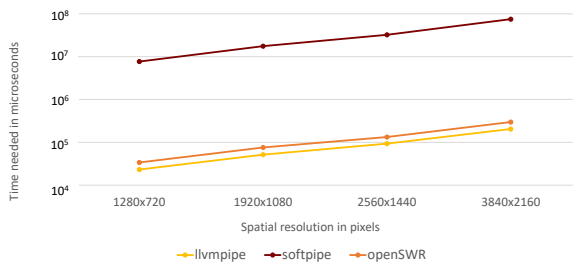


Figure 1: Runtime performance comparison in microseconds of software drivers (LLVMpipe vs. Softpipe vs. OpenSWR) using morphological closing operation with a kernel size of three (logarithmic scale).

Based on this analysis, the LLVMpipe backend for SWR is chosen for the image-processing techniques because the supported features are sufficient and the performance is compared to the other drivers the best one for this task.

3.2 Deployment Process

Similar to approaches used for real-time 3D rendering [24], the deployment process basically comprises the following steps that can be managed using an automatic approach. At first, LLVM, is compiled and linked. Subsequently, Mesa3D with LLVMpipe backend is compiled and linked. After that, the Mesa3D OpenGL dynamic linked library for Windows targets (`opengl32.dll`) or shared library for UNIX-like systems (`libGL.so`) is placed in the directory next to the executable.

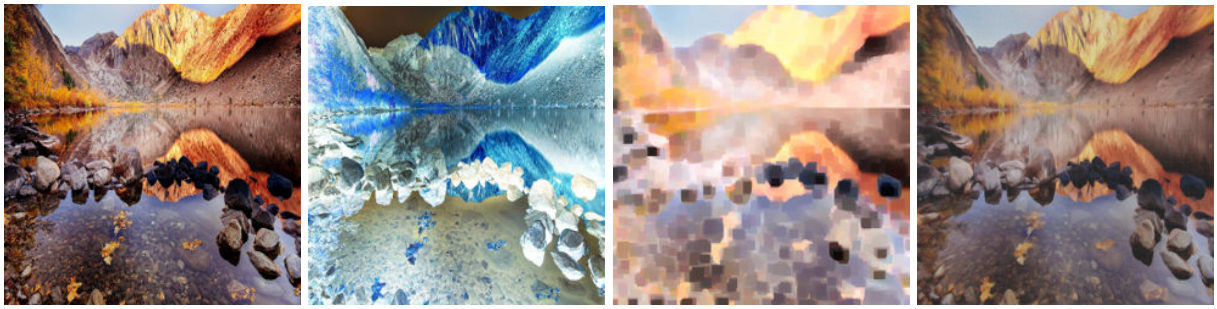
Following to that, OpenGL-specific application calls are resolved using the LLVMpipe driver. Finally, for patching the application, a docker image [1] is prepared and deployed to the respective cloud-service providers.

4 PERFORMANCE EVALUATION

This section evaluates, compares, and discusses the runtime performance of the service-based image processor deployed to (1) a dedicated GPU server and (2) different standard cloud-computing platforms.

4.1 Test Data and Processing Techniques

Different image resolutions were tested with different image-processing techniques to estimate the performance of software rendering regarding the spatial resolution of an image as well as the complexity of processing techniques. The following common resolutions in pixels were chosen: 1280×720 (HD), 1920×1080 (FHD), 2560×1440 (QHD), and 3840×2160 (4K).



(a) Original. (b) Color invert. (c) Morphological closing. (d) Oilpaint.

Figure 2: Exemplary results of different image-processing techniques (b) – (d) that are used for performance evaluation and comparison applied to an input image (a).

In addition to various spatial resolutions, different image processing techniques (Figure 2) have been tested in order to cover a broad spectrum and obtain variant estimates on software rendering performance and behavior with respect to different types of processing tasks:

Color Invert: This operation is pixel-based and inverts the color value of every pixel. This requires only a single sampling operation followed by an absolute value of a single arithmetic operation (Figure 2b).

Morphological Closing: This operation is neighborhood-based. It's a morphological operation and comprises a dilation followed by an erosion. Separated kernels are used. Three different kernel sizes were tested: 3, 14, and 90 (Figure 2c).

Oilpaint Abstraction: This multi-pass processing technique [33] consists mainly of color palette extraction, colorization, luminance quantization, edge detection, flow-field computation, smoothing and blending techniques (Figure 2d).

4.2 Overview of Test Systems

We evaluate the run-time performance of above processing techniques applied to the test input data using servers provided by different vendors. Table 2 (next page) shows an overview of the hardware and software specifications of the test systems:

Dedicated GPU Server: This server has dedicated graphics hardware and was used to measure the competitive performance of GPU-based processing as well as processing using SWR.

Amazon EC2 t2.large: This is a general purpose AWS EC2 instance of the type t2.large.

Amazon EC2 c4.4xlarge: This is a compute-optimized AWS EC2 instance of the type c4.4xlarge.

Amazon EC2 c5.18xlarge: This is highly compute-optimized AWS EC2 instance of the type c5.18xlarge. It has additional RAM and an increased number of virtual CPUs (vCPUs) compared to the c4.4xlarge instance.

4.3 Test Setup

In order to obtain the performance measurements, two docker containers were launched. As shown in Figure 3, one container runs the actual image processor instance, while the other container exposes a REST interface for communication with the client. It consists of a NodeJS [34] server that communicates with the image processor instance via WebSockets [29]. The image processor itself had a *profiler* implementation that could also be configured using the REST interface and performed measurements whenever an image was processed. Also the obtained performance results could be queried using this interface.

After the docker containers were launched, the profiler was configured to measure only the duration of the actual processing of an image – not the setup of the operation or the transmission of the image. In a real-world scenario the transmission and loading times are of course relevant and should be taken into consideration but these are not the focus of this work. The profiler uses OpenGL query objects to obtain the measurements of the actual processing time and not only the time spend to call the asynchronous OpenGL requests.

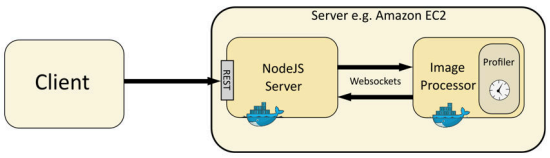


Figure 3: Setup and deployment for service-based performance testing. Via REST interface, a client communicates with the image-processing service that controls an image processing instance using WebSockets.

Table 2: Overview of cloud-computing provider used for performance evaluation (all 64 bit architecture).

Aspect	Dedicated GPU Server	AWS EC2 t2.large	AWS EC2 c4.4xlarge	AWS EC2 c5.18xlarge
Processor	Xeon E5-2637 v4, 3.5 GHz	Xeon, 3.0 GHz	Xeon E5-2666 v3, 2.9 GHz	Xeon Platinum, 3.0 GHz
Cores/Threads	8 cores	2 vCPUs	16 vCPUs	72 vCPUs
Memory	64 GB	8 GB	30 GB	144 GB
GPU	NVIDIA Quadro M6000 24 GB	None	None	None

When using software rendering, this obviously makes no difference because everything that would be processed by a GPU is processed by the CPU itself. For every combination of resolution and processing technique six identical image processing requests were sent and the results of the measurements queried and saved.

4.4 Test Results

From the six measurements obtained for every combination of image resolution and processing technique, the first one was ignored and the average of the remaining ones was calculated. This was done due to the fact that the first of these six measurements usually was significantly higher than the remaining (ranging from a factor of 1.2 up to a factor of over 1000) because OpenGL optimizes after the first run of processing an image with a specific processing technique. This, of course, is an important aspect that should be taken into account when deploying a service-based image processing service for multiple users that probably want to process images with different resolutions and different processing techniques.

Figure 4 shows the results of the measurements. The first four charts show the absolute processing time in microseconds for a specific resolution and processing technique comparing GPU-based rendering (solid lines) to SWR (dotted lines) on the different machines. All charts are shown using a logarithmic scale.

Figure 5 shows the *speed factor* for software rendering on each of the four test machines, i.e., how much slower software rendering was comparedto GPU-based rendering. The factor was calculated by dividing the duration of GPU-based processing by the duration using software rendering. In the case that SWR was faster, its duration of was divided by the duration of the GPU-based approach. The result then was negated to emphasize that software rendering was actually faster. Note that in case of the color invert, the values are negative, i.e., for this processing technique SWR was actually faster than GPU-based rendering.

4.5 Discussion of Test Results

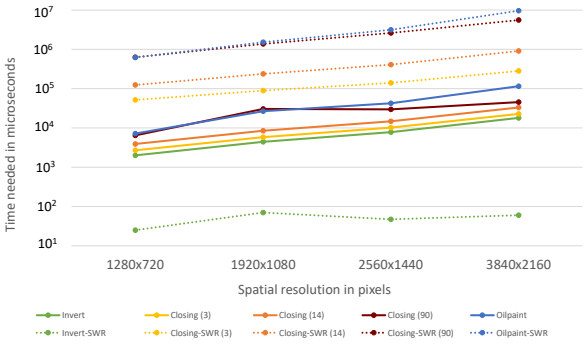
At first, the processing duration of software rendering compared to GPU-based rendering is discussed. Subsequently, we discuss the computed speed factors.

It can be observed that SWR generally is significantly slower than GPU-based rendering, but the relations between the different processing techniques and image resolutions for SWR are similar to GPU-based rendering. For example, for both SWR and GPU-based rendering, oilpaint abstraction, and morphological closing with kernel size 90 have a similar processing duration for smaller image resolutions, but for higher image resolutions the oilpaint abstraction exhibit high runtimes. This shows that SWR introduces an impact to the processing performance, but it does not change how the different processing techniques and image resolutions compare to each other regarding performance, e.g., which processing technique has the shortest duration for a specific resolution. While SWR is slower for most of the processing techniques, interestingly for color invert this is not the case. There are two possible explanations for this: first of all we only tested GPU-based processing using one specific GPU. It could be that on other GPUs color invert would be faster. Another explanation could be that transferring the image to the GPU takes longer compared to perform an invert on the CPU, even if the CPU cannot parallelize this similar to the GPU [15].

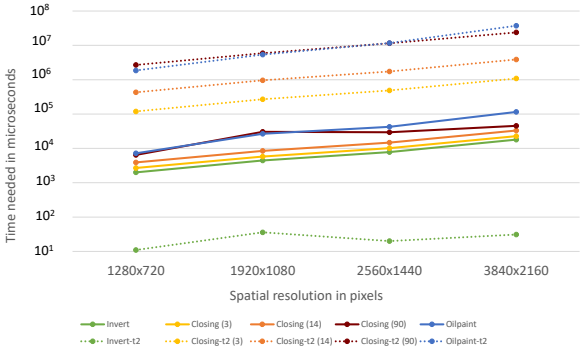
The measurements show that an approach for reducing the processing time when using SWR is to increase the number of vCPUs. Figure 6 shows the speed increase by increasing vCPUs based on the taken measurements for morphological closing with kernel size of 14 on a Full High Definition (FHD) image on the three Amazon EC2 instances. This speed factor is computed by dividing the processing duration using the EC2 t2.large instance (which has two vCPUs) by the processing duration if using additional vCPUs. It can be observed that the speed does not increase linearly with an increasing amount of vCPUs. For a final conclusion, more measurements on different machines would be required but there probably exists an upper bound to which the speed factor can be pushed.

Regarding the speed factor for SWR compared to GPU-based rendering, it can be observed that for single or multi-pass processing techniques based on pixel-based or neighborhood-sampling of small kernel-sizes the factor remains stable. This allows for assessing the performance impact when using SWR for these processing techniques. However, for complex processing techniques it can be observed that with increasing image resolution, GPU-based rendering performance is superior compared to SWR.

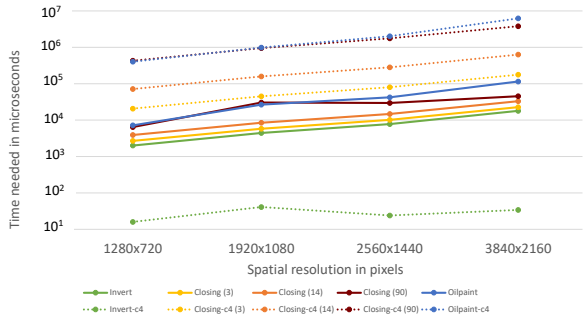
Thus, for more complex processing techniques, the performance impact depends strongly on the spatial res-



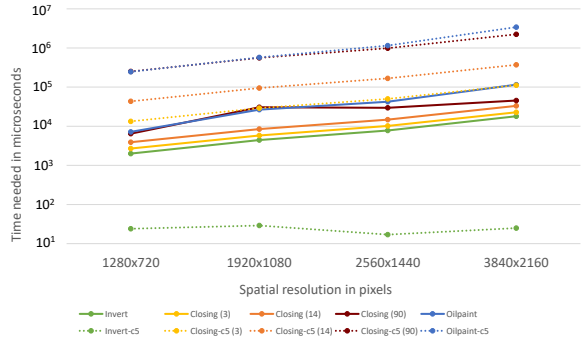
(a) GPU vs. SWR on dedicated server.



(b) GPU vs. AWS EC2 t2.large.

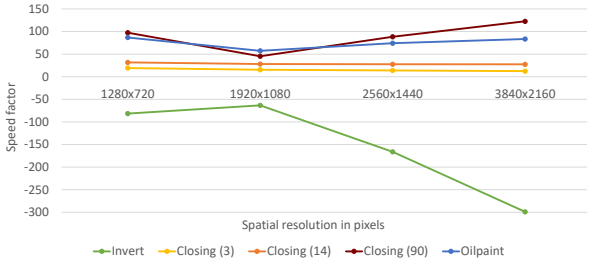


(c) GPU vs. AWS EC2 c4.4x large.

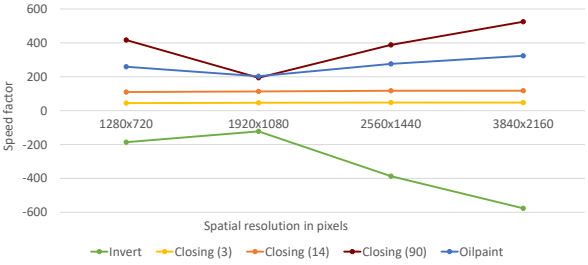


(d) GPU vs. AWS EC2 c5.18xlarge.

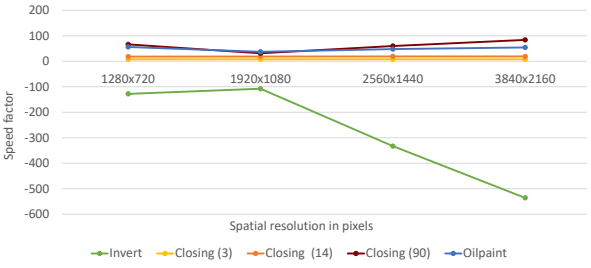
Figure 4: Comparing processing duration of GPU-based rendering to software rendering on different test systems.



(a) SWR on dedicated server.



(b) AWS EC2 t2.large.



(c) AWS EC2 c4.4xlarge.

(d) AWS EC2 c5.18xlarge.

Figure 5: Speed factor of software rendering compared to GPU-based rendering for different machines.

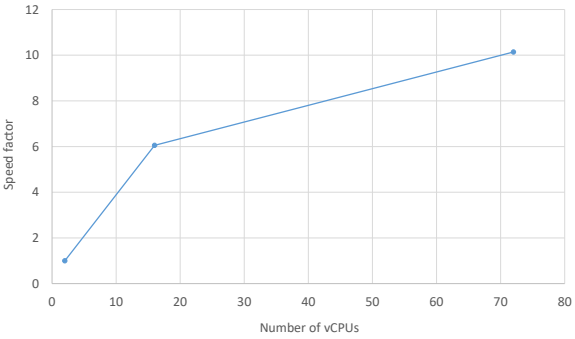


Figure 6: Speed factor for a specific amount of vCPUs compared to two vCPUs.

olution of the input image. It's interesting that for complex processing techniques (e.g., oilpaint abstraction and morphological closing with kernel size of 90), the speed factor for SWR on all test machines when processing a FHD image is noticeable lower, which is reflected by the bend in the graph. The origin of this bend lies in a higher processing duration for FHD images compared to Quad High Definition (QHD) images when using GPU-based rendering.

It should be mentioned that a FHD image has 2.25 times more pixels than a High Definition (HD) image and a 4K image also has 2.25 times more pixels than a FHD image, while a QHD image only has approx. 1.7 times more pixels than a FHD image. This means that regarding that number-of-pixels the measurements results for QHD images would be closer to the FHD images, compressing the graph in the horizontal direction. This of course does not explain why the processing duration for QHD is lower than the processing time for FHD. For this, we found no sufficient explanation. The reason may lie within the specific GPU we used for testing.

5 CONCLUSIONS

This paper presents the results of feasibility study and rendering performance evaluation of service-based image processing techniques based on OpenGL software rendering. It compares the rendering performance results with a dedicated GPU-accelerated service deployment. The results show a significant negative performance of software rendering compared to GPU-accelerated processing. However and to a certain extend, this limitations can be attenuated by increasing the number of vCPUs/Threads.

Further, the performance of software rendering compared to GPU-based rendering strongly depends on the implementation complexity of the processing technique, i.e., for less complex processing operations the speed factor remains stable. Furthermore, for less complex processing techniques using pixel-based sampling, software rendering can be faster.

ACKNOWLEDGMENTS

We would like to thank the anonymous reviewers for their valuable feedback. This work was funded by the Federal Ministry of Education and Research (BMBF), Germany, for the AVA project 01IS15041.

REFERENCES

[1] Carl Boettiger. An introduction to docker for reproducible research. *SIGOPS Oper. Syst. Rev.*, 49(1):71–79, January 2015.

[2] Aleksey Bragin, Alexander Dubanov, and Alexander Rechitskiy. User Interface Design for a Web-based Image Processing and Analysis System. In C. Wernhard S. Hölldobler, A. Malikov, editor, *YSIP2 - Proceedings of the Second Young Scientist's International Workshop on Trends in Information Processing*, Dombai, Russian Federation, 2017. CEUR.

[3] Tobias Dürschmid, Maximilian Söchting, Amir Semmo, Matthias Trapp, and Jürgen Döllner. Prosumerfx: Mobile design of image stylization components. In *SIGGRAPH Asia 2017 Mobile Graphics & Interactive Applications*, SA '17, pages 1:1–1:8, New York, NY, USA, 2017. ACM.

[4] Thomas Erl. *Service-Oriented Architecture: Concepts, Technology, and Design*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2005.

[5] Romain Failliot, Tobias Droste, and Robin McCorkell. The OpenGL vs Mesa matrix. `https://mesamatrix.net`. Accessed: 2019-04-26.

[6] Zeev Farbman, Raanan Fattal, Dani Lischinski, and Richard Szeliski. Edge-preserving decompositions for multi-scale tone and detail manipulation. *ACM Transactions on Graphics*, 27(3):67:1–67:10, 2008.

[7] Leon A. Gatys, Alexander S. Ecker, and Matthias Bethge. Image style transfer using convolutional neural networks. In *Proc. Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2414–2423, Los Alamitos, 2016. IEEE Computer Society.

[8] Kengo Hayashi, Naohisa Sakamoto, Jorji Nonaka, Motohiko Mastuda, and Fumiyoshi Shoji. An In-Situ Visualization Approach for the K computer using Mesa 3D and KVS. In *ISC Workshop on In-Situ Visualization 2018 (WOIV 2018)*, 2018.

[9] Kaiming He, Jian Sun, and Xiaoou Tang. Guided image filtering. In *Proc. European Conference on Computer Vision (ECCV)*, pages 1–14. Springer, 2010.

[10] Alexander Jungmann and Bernd Kleinjohann. Automatic composition of service-based image processing applications. In *Proc. IEEE Interna-*

tional Conference on Services Computing (SCC), pages 106–113. IEEE Computer Society, 2016.

[11] Michael Kass and Justin Solomon. Smoothed local histogram filters. *ACM Transactions on Graphics*, 29(4):100:1–100:10, 2010.

[12] David Koller, Michael Turitzin, Marc Levoy, Marco Tarini, Giuseppe Croccia, Paolo Cignoni, and Roberto Scopigno. Protected Interactive 3D Graphics via Remote Rendering. *ACM Trans. Graph.*, 23(3):695–703, August 2004.

[13] Jan Eric Kyprianidis, John Collomosse, Tinghuai Wang, and Tobias Isenberg. State of the "art": A taxonomy of artistic stylization techniques for images and video. *IEEE Transactions on Visualization and Computer Graphics*, 19(5):866–885, 2013.

[14] Samuli Laine and Tero Karras. High-performance software rasterization on gpus. In *Proceedings of the ACM SIGGRAPH Symposium on High Performance Graphics*, HPG '11, pages 79–88, New York, NY, USA, 2011. ACM.

[15] Victor W. Lee, Changkyu Kim, Jatin Chhugani, Michael Deisher, Daehyun Kim, Anthony D. Nguyen, Nadathur Satish, Mikhail Smelyanskiy, Srinivas Chennupaty, Per Hammarlund, Ronak Singhal, and Pradeep Dubey. Debunking the 100X GPU vs. CPU Myth: An Evaluation of Throughput Computing on CPU and GPU. In *Proceedings of the 37th Annual International Symposium on Computer Architecture*, ISCA '10, pages 451–460, New York, NY, USA, 2010. ACM.

[16] Angel Lagares Lemos, Florian Daniel, and Boualem Benatallah. Web service composition: A survey of techniques and tools. *ACM Computing Surveys*, 48(3):33:1–33:41, 2015.

[17] The LLVM Compiler Infrastructure. `http://llvm.org`. Accessed: 2019-04-26.

[18] Kwan-Liu Ma and Steven Parker. Massively Parallel Software Rendering for Visualizing Large-Scale Data Sets. *IEEE Computer Graphics and Applications*, 21(4):72–83, July 2001.

[19] The Mesa 3D Graphics Library. `https://www.mesa3d.org`. Accessed: 2019-04-26.

[20] Peter Mileff and Judit Dudra. Efficient 2D Software Rendering. *Production Systems and Information Engineering*, 6(2):55–66, May 2012.

[21] Peter Mileff and Judit Dudra. Modern Software Rendering. *Production Systems and Information Engineering*, 6(2):99–110, May 2013.

[22] Himadri Nath Moulick and Moumita Ghosh. Medical image processing using a service oriented architecture and distributed environment. *American Journal of Engineering Research (AJER)*, 2(10):52–62, 2013.

[23] Matthias Mueller and Benjamin Pross. *OGC WPS 2.0.2 Interface Standard*. Open Geospatial Consortium, 2015. http://docs.opengeospatial.org/is/14-065/14-065.html.

[24] Christopher Niederauer, Mike Houston, Maneesh Agrawala, and Greg Humphreys. Non-invasive interactive visualization of dynamic architectural environments. In *Proceedings of the 2003 Symposium on Interactive 3D Graphics*, I3D '03, pages 55–58, New York, NY, USA, 2003. ACM.

[25] Mike P. Papazoglou and Willem-Jan Heuvel. Service oriented architectures: Approaches, technologies and research issues. *The VLDB Journal*, 16(3):389–415, 2007.

[26] Chris Peltz. Web services orchestration and choreography. *Computer*, 36(10):46–52, October 2003.

[27] Pietro Perona and Jitendra Malik. Scale-space and edge detection using anisotropic diffusion. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(7):629–639, 1990.

[28] Ricardo Queirós and Alberto Simões. SOS - Simple Orchestration of Services. In Ricardo Queirós, Mário Pinto, Alberto Simões, José Paulo Leal, and Maria João Varanda, editors, *6th Symposium on Languages, Applications and Technologies (SLATE 2017)*, volume 56 of *OpenAccess Series in Informatics (OASIcs)*, pages 13:1–13:8, Dagstuhl, Germany, 2017. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

[29] Marvin Richter, Maximilian Söchting, Amir Semmo, Jürgen Döllner, and Matthias Trapp. Service-based Processing and Provisioning of Image-Abstraction Techniques. In *Proceedings of the 26th International Conference on Computer Graphics, Visualization and Computer Vision*, Proceedings International Conference on Computer Graphics, Visualization and Computer Vision (WSCG), pages 97–106, 2018.

[30] Timothy Rowley. Software Rasterizer (SWR). In *Presented at the Intel HPC Developers Conference at SC14*, 2014.

[31] Mark Segal and Kurt Akeley. *The OpenGL Graphics System: A Specification (Version 4.6 (Core Profile) - May 14, 2018)*. The Khronos Group Inc., May 2018.

[32] Mark Segal and Kurt Akeley. *Vulkan 1.0.107: A Specification*. The Khronos Group Inc., April 2019.

[33] Amir Semmo, Daniel Limberger, Jan Eric Kyprianidis, and Jürgen Döllner. Image stylization by interactive oil paint filtering. *Computers & Graphics*, 55:157–171, 2016.

[34] Lambert M. Surhone, Mariam T. Tennoe, and Susan F. Henssonow. *Node.Js*. Betascript Publishing, Mauritius, 2010.

[35] Carlo Tomasi and Roberto Manduchi. Bilateral filtering for gray and color images. In *Proc. International Conference on Computer Vision (ICCV)*, pages 839–846. IEEE, 1998.

[36] Mircea-Florin Vaida, Valeriu Todica, and Marcel Cremene. Service oriented architecture for medical image processing. *International Journal of Computer Assisted Radiology and Surgery*, 3(3):363–369, 2008.

[37] Joachim Weickert. *Anisotropic diffusion in image processing*, volume 1. Teubner Stuttgart, 1998.

[38] Robert P. Winkler and Chris Schlesiger. Image processing rest web services. Technical Report ARL-TR-6393, Army Research Laboraty, Adelphi, MD 20783-119, 2013.

[39] Holger Winnemöller, Sven C. Olsen, and Bruce Gooch. Real-time video abstraction. *ACM Transactions on Graphics*, 25(3):1221–1226, 2006.

[40] M. Würsch, R. Ingold, and M. Liwicki. Sdk reinvented: Document image analysis methods as restful web services. In *2016 12th IAPR Workshop on Document Analysis Systems (DAS)*, pages 90–95, 2016.

[41] Marcel Würsch, Rolf Ingold, and Marcus Liwicki. Divaservices - a restful web service for document image analysis methods. *Digital Scholarship in the Humanities*, 32(1):i150–i156, 2017.

[42] Xiaoxia Yang. Remotely sensed image processing service automatic composition. State Key Laboratory of Information Engineering in Surveying, Mapping and Remote Sensing, Wuhan University, 2009.

[43] Rong Yuan, Ming Luo, Zhi Sun, Shuyue Shi, Peng Xiao, and Qingguo Xie. Rayplus: a web-based platform for medical image processing. *J. Digital Imaging*, 30(2):197–203, 2017.